(English Language Translation)

NOTICE OF REASONS FOR REJECTION

Patent Application No. : H11-319570
Drafting Date: June 28, 2004 (Mailed on July 6, 2004)
Paten office Examiner : Kazushige Goto
Agent of the Applicant : Fujio SASAJIMA
Applied Articles : Article 29, Paragraph 2, Article 36

It is deemed that this application should be rejected for the following reasons. In cases where there is any opinion on this action, it is requested that a written Comment should be submitted within 60 days from the date on which this notification was dispatched.

REASONS

A. The invention of this application recited in the following claims cannot be patented in accordance with Article 29, Paragraph 2 of the Patent Law because it is deemed that it could have easily been accomplished prior to the application on the basis of the invention described in the publication(s) mentioned below, which are distributed in Japan or foreign countries prior to filing of the application, or available to the public via electric telecommunication lines by a person having common knowledge in the art of field to which the invention belongs.

B. This application does not comply with the requirements in accordance with Article 36, Paragraph 6 of the Patent Law on the points mentioned below.

REMARKS

(Please refer to the following list of references cited.)

<With regard to the Reasons for Rejection A>
RE: Claims 1 and 4
Cited References: 1
Remarks:

In a method of managing thread private data, the cited reference 1 discloses the construction in which a stack peculiar to each of the multiple threads is allocated to respective threads when the threads are generated.

When performing a procedure call, because a parameter regarding the procedure is generated as a frame on the stack, the "stack" described in the cited reference 1 is equal

to a "dynamically and thread-piece-wise allocated interface area" of the invention recited in the above-described claims.

RE: Claims 2, 3, 5, and 6
Cited References: 1
Remarks:

Paragraph [0020] of the cited reference 1 discloses the stack managed by each thread is performed when threads are generated, the aforementioned point is equal to "code generating means" and "code converting means" of the invention recited in the above-described claims.

<With regard to the Reasons for Rejection B>

A "compiler device" of the invention recited in claim 1 may be deemed as the "product" invention, however, it is considered that the construction understood from claim 1 is substantially the "method" invention and accordingly, the construction as the "product" invention is unclear.

List of References Cited

1. Japanese Unexamined Patent Publication No. H05-204656

For the claims other than the claims specified in this notice of reasons for rejection, no reason for rejection is found at present. If any reasons for rejection are found later, it will be notified.

Particulars of Results of Related Art Search

Technical Field of Search: IPC 7th Edition G06F9/44

Related Art Information

Japanese Unexamined Patent Publication No. H02-56640

These three references were not relied upon in the claim rejection under Paragraph 2, Article 29 of the Patent Law.

整理番号 9951141

発送番号 239653 1/
発送日 平成16年 7月 6日

拒絶理由通知書

99-51141
P199-0324

特許出願の番号	平成11年 特許願 第319570号
起案日	平成16年 6月28日 16.9.04
特許庁審査官	後藤 和茂 9463 5B00
特許出願人代理人	笹島 富二雄 様
適用条文	第29条第2項、第36条

この出願は、次の理由によって拒絶をすべきものである。これについて意見があれば、この通知書の発送の日から60日以内に意見書を提出して下さい。

理 由

A．この出願の下記の請求項に係る発明は、その出願前日本国内又は外国において頒布された下記の刊行物に記載された発明に基いて、その出願前にその発明の属する技術の分野における通常の知識を有する者が容易に発明をすることができたものであるから、特許法第29条第2項の規定により特許を受けることができない。

B．この出願は、特許請求の範囲の記載が下記の点で、特許法第36条第6項第2号に規定する要件を満たしていない。

記 (引用文献等については引用文献等一覧参照)

＜理由Aについて＞

請求項1、4について
引用文献1
備考：
引用文献1には、スレッド固有データ保持方法において、スレッド生成時にそれぞれのスレッドに固有のスタックを割り当てる構成が記載されている。手続呼び出しを行う際には、スタック上に手続に関するパラメータをフレームとして生成するものであるから、引用文献1に記載の「スタック」は、上記請求項に係る発明の「スレッド単位毎に動的に確保されたインターフェース領域」に相当する。

請求項2、3、5、6について
引用文献1

・整理番号 ９９５１１４１ 発送番号 ２３９６５３ 2/E
発送日 平成１６年 ７月 ６日

備考：

引用文献段落番号【００２０】には、各スレッドが保持するスタックは、スレッド生成時に行う点が記載されており、前記点は上記請求項に係る発明の「コード生成手段」及び「コード変換手段」に相当する。

＜理由Ｂについて＞

請求項１に係る発明の「コンパイラ装置」において、「物」の発明と認められるが、請求項１から把握される構成は実質的に「方法」の発明であり、「物」の発明としての構成が不明瞭である。

引 用 文 献 等 一 覧

１．特開平５－２０４６５６号公報

この拒絶理由通知書中で指摘した請求項以外の請求項に係る発明については、現時点では、拒絶の理由を発見しない。拒絶の理由が新たに発見された場合には拒絶の理由が通知される。

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

先行技術文献調査結果の記録

・調査した分野 ＩＰＣ第７版 Ｇ０６Ｆ９／４４
・先行技術文献 特開平２－５６６４０号公報

この先行技術文献調査結果の記録は、拒絶理由を構成するものではない。

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平5－204656

(43)公開日　平成５年(1993)８月13日

(51)Int.Cl.[5]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　9/45				
		9292－5Ｂ	Ｇ０６Ｆ　9/44　３２２　Ａ	

審査請求　未請求　請求項の数２(全 12 頁)

(21)出願番号　特願平4－65772

(22)出願日　平成４年(1992)３月24日

(31)優先権主張番号　特願平3－339572

(32)優先日　平３(1991)11月30日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地

(72)発明者　白木原　敏雄
神奈川県川崎市幸区小向東芝町１番地　株式会社東芝総合研究所内

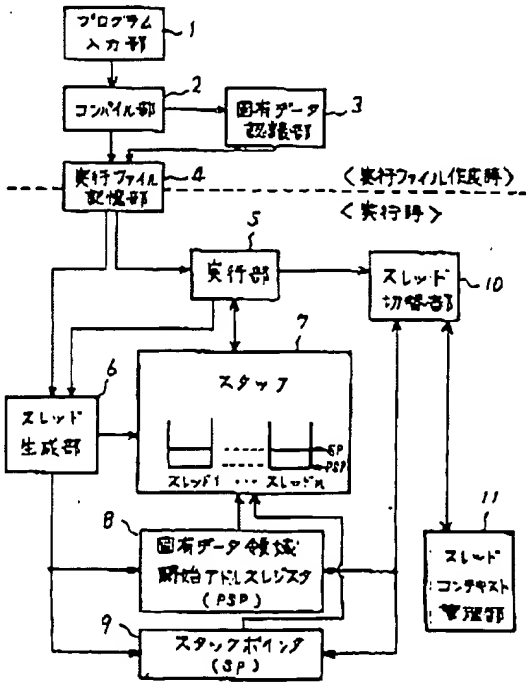
(74)代理人　弁理士　則近　憲佑

(54)【発明の名称】　スレッド固有データ保持方法

(57)【要約】

【目的】　データ空間を共有するスレッドが大域的に宣言された変数を固有に保持する方法の提供を目的とする。

【構成】　アドレス空間を定義するタスク内に実行主体のスレッドが複数存在するプログラムを解析して複数スレッド毎に固有な大域データの抽出、領域の大きさおよび領域中の固有データのオフセットの決定、そして、実行時に設定される領域のベースアドレスとオフセットでアクセスする実行命令の生成を固有データ認識部３で行い、スレッド生成時に大域データ領域の大きさをもとにスレッドのスタックに割り当てる固有データ領域の開始アドレスの設定をスレッド生成部６で行い、前記大域データを割り付ける領域をスタック７中に確保するとともに上記領域の開始アドレスをプログラム実行中にスレッドごとに固有データ領域開始アドレスレジスタ８に保持するようにしている。



【特許請求の範囲】

【請求項１】　アドレス空間を定義するタスク内に実行主体のスレッドが複数存在するよう電子計算機を動作させるプログラムを入力し、

この複数のスレッドから大域的にアクセス可能でかつ各スレッド毎に固有に保持すべきデータを、入力した前記プログラムから抽出し、

各スレッド毎に保有するスタックにそれぞれ、抽出した前記データを割り当てる領域を確保し、

この確保した領域の基準となるアドレスを前記各スレッド毎に保持し、

前記データに対するアクセスは、保持した前記アドレスに基づいて行うことを特徴とするスレッド固有データ保持方法。

【請求項２】　アドレス空間を定義するタクス内に実行主体のスレッドが複数存在するよう電子計算機を動作させるプログラムを入力し、

この複数のスレッドから大域的にアクセス可能でかつ各スレッド毎に固有に保持すべきデータを複数個、入力した前記プログラムから抽出し、

この抽出した複数個のデータを割り当てるのに要する領域の大きさと、この領域内での各データの位置とを決定し、

この決定した位置に基づいて各データのアクセス命令を生成しておき、

前記スレッドを生成する際に、決定した前記大きさに基づいてこのスレッドが保有するスタックに前記データを割り当てる領域を確保し、

この確保した領域の基準となるアドレスを、生成した前記スレッドが他のスレッドに切り替えられるかあるいは消滅するまでの間保持し、

前記データに対してアクセスする際には、生成した前記アクセス命令と保持した前記アドレスとに基づいてアクセスを行うことを特徴とするスレッド固有データ保持方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、電子計算機の並列処理において、アドレス空間を共有しスタックのみを持つスレッドが、大域的に宣言されたデータを固有に保持するスレッド固有データ保持方法に関する。

【０００２】

【従来の技術】従来、並列処理を効率よく実現するためプロセスという概念をタスクとスレッドという２つの概念に分解するモデルが提案され、実現されている。

【０００３】ここで、タスクとはアドレス空間などの静的な実行環境を定義するものであり、スレッドとは、スタックやプログラムカウンタなどの動的な実行環境を定義するものである。

【０００４】このようなモデルでは、１つのタスク内に複数のスレッドが同時に存在することが可能である。このことは、１つのタスク内の同じデータを用いて、複数の処理を同時に行うことができることを意味する。また、タスクとスレッドのモデルは、単純にプロセスによる並列処理を行う場合と比べて、次のような利点がある。

・スレッド間の通信は、共有データを介して行うために高速である。

・スレッドはプロセスに比べて独自に持つ資源が少ないため、生成・消滅のオーバヘッドが少ない。

【０００５】また、このようなタスクとスレッドのモデルの場合、スレッド間でアドレス空間を共有しているため、大域データはすべてのスレッドから共有されるようになる。

【０００６】例えば、Ｃ言語のプログラムで使用されるデータは、大域変数、Ａｕｔｏ変数の２つに大別でき、このうち大域変数は、複数の関数からアクセスできるもので通常データ空間に割り当てられ、上述の大域データに相当しスレッド間で共有されるものである。一方、Ａｕｔｏ変数は、関数内で宣言され、宣言された関数内で有効なもので、通常スタックに割り当てられる。

【０００７】さて、１つのタスク中に複数のスレッドが存在するマルチスレッド環境においては、大域的に宣言されたデータをスレッド毎に保持する必要がある。スレッド固有の大域データの必要な例として、マルチスレッドサーバを次に示す。

【０００８】ここでのマルチスレッドサーバは、同じサービスを行うスレッドが１つのタスク内に複数存在し、他のサーバからのサービス要求を同時に複数処理するもので、次のようなプログラムにより実現される。

```
ｓｔｒｕｃｔ　ｍｓｇ　ｍ；
ｇｅｔ＿ｊｏｂ（）｛
・・・
ｒｅｃｅｉｖｅ（&ｍ）；
・・・
ｊｏｂ（）；
｝
```

【０００９】この場合、各スレッドは外部からのサービス要求を受けるために、関数ｇｅｔ＿ｊｏｂを呼び出し、その内部で関数ｒｅｃｅｉｖｅを呼び出し、メッセージの内容を変数ｍで受けとる。このメッセージ変数ｍは通常複数の関数で処理されるため、大域的に扱う必要がある。しかし、マルチスレッド環境では、大域的に宣言されたデータは、スレッド間で共有されるため、上記のようなプログラムでは、メッセージが破壊される可能性がある。例えば、複数のサービス要求が来た場合、それらの要求内容はメッセージ変数ｍに格納されるが、メッセージ変数ｍはスレッド間で共有しているため、後から来たメッセージの内容が上書きされ、前のメッセージが破壊されるおそれがある。このため、大域データをス

レッド固有に保持するための特別なデータ管理方式が必要になっている。従来、提案されてきた方法として、以下の2つの方法が知られている。

1．動的にスタックに割り当てる方法

【0010】この方法は、各スレッドが実行する関数内でＡｕｔｏ変数として宣言することにより、スレッド固有データをスタックに割り当てる。しかし、その関数を呼び出した他の関数内で、その変数を参照する場合、Ａｕｔｏ変数は宣言された関数外では直接参照できないため、変数のアドレスを関数引数として渡す必要が生じる。この方法によるとプログラム例として図９に示すものがある。

【0011】この場合、各スレッドは、その生成後、関数ｓｔａｒｔから実行を開始する。また、スレッド毎に固有データｓｈａｒｅｄ＿ｄａｔａを保持するために、関数ｓｔａｒｔ内で宣言し、スタックに割り当てる。関数ｓｔａｒｔから関数ｆｕｎｃ１、ｆｕｎｃ２を呼び出す場合、それぞれの関数内では固有データｓｈａｒｅｄ＿ｄａｔａが参照できないため、そのアドレスを関数引数として渡す必要がある。

2．変数をスレッド毎に拡張し、スレッドＩＤでアクセスする方法。

【0012】この方法は、固有に保持すべきデータを、スレッド数の要素を持つ大域的な配列として宣言し、スレッド毎に一意に与えられたＩＤによりアクセスする方法である。このような方法では、スレッドＩＤがＯＳレベルでサポートされていない場合、スレッドＩＤ自体を上述の方法１で保持する必要があり、また、スレッドの数が動的に変化する場合には対応できない。この方法によるプログラム例として図１０に示すものがある。

【0013】この場合、固有データｓｈａｒｅｄ＿ｄａｔａは、大域的な配列として宣言されている。その要素数は、定数ＴＨＲＥＡＤ＿ＮＵＭで定義している。また、固有データのアクセスはスレッドＩＤであるＴＨＲＥＡＤ＿ＳＥＬＦを使用して行われる。

【0014】この例では、配列の要素数は静的にＴＨＲＥＡＤ＿ＮＵＭまでに定まってしまい、スレッドの数に制限が生じる。また、固有データへアクセスする場合には、スレッドＩＤであるＴＨＲＥＡＤ＿ＳＥＬＦのＯＳレベルでのサポートが必要である。

【0015】

【発明が解決しようとする課題】このように従来のスレッドが大域データを固有に持つ方法において、関数内でＡｕｔｏ変数として宣言する方法では、他の関数を呼び出す場合に固有データのアドレスを関引引数として渡す必要があるため、固有データの操作を行なう関数の呼び出しでは、アドレスを指定する引数を含み、プログラムが繁雑になってしまう。また、呼び出された関数では、固有データのアドレスをスタックに保存する必要があり、さらに、固有データへのアクセスはアドレスによる間接アクセスしか行なえないため、実行効率が低下することになる。

【0016】一方、大域的な配列として宣言する方法では、スレッドＩＤが必要であり、さらにスレッドの動的な生成・消滅に容易に対応できない。また、スレッドの動的生成・消滅に対応するためには、スレッドＩＤをハッシングして配列を引くような構造にしなくてはならないために実行効率が低下することになる。また、スレッドの最大個数は、配列の要素数に制限されるようにもなる。

【0017】このように、従来のものでは、スレッド固有の大域データを実現するために、プログラムの書き易さや実行効率を犠牲にしなければならないなどの問題点を有している。

【0018】本発明は、上記事情に鑑みてなされたもので、スレッド固有データを大域変数と同様に宣言することができ、また、スレッド数の動的な変化にも対応できるスレッド固有データ保持方法を提供することを目的とする。

【0019】

【課題を解決するための手段】本発明に係るスレッド固有データ保持方法は、アドレス空間を定義するタスク内に実行主体のスレッドが複数存在するよう電子計算機を動作させるプログラムを入力し、このプログラムから、複数のスレッドから大域的にアクセス可能でかつ各スレッド毎に固有に保持すべきデータを抽出し、各スレッド毎に保有するスタックのそれぞれに抽出した前記データを割り当てる領域を確保し、この確保した領域の基準となるアドレスを前記各スレッド毎に保持し、前記データに対するアクセスは、保持した前記アドレスに基づいて行うことを特徴とするものである。

【0020】複数のスレッドから大域的にアクセス可能でかつ各スレッド毎に固有に保持すべきデータが前記プログラム中に複数個ある場合には、この複数個のデータを割り当てるのに要する領域の大きさと、この領域内での各データの位置とを決定し、この決定した位置に基づいて各データのアクセス命令を生成しておき、各スレッドが保有するスタックにおける領域確保は、決定した前記大きさに基づいて行い、前記データに対するアクセスは、生成した前記アクセス命令と保持した前記アドレスとに基づいて行うものである。また、領域確保と基準となるアドレス保持はスレッド生成時に行い、データへのアクセスの際に用いるアドレスは、実行中のスレッドについて保持されているものとするというものである。

【0021】

【作用】本発明によれば、複数のスレッドから可視の大域データであって且つ各スレッド毎に固有に保持すべきデータであるものは、その旨をプログラム中で宣言しておけば、このような宣言をされたデータについてはそのための領域を各スレッド毎にスタックに確保し、その領

域の基準となるアドレスを保持し、保持したアドレスに基づいてデータへのアクセスを行うため、このデータは各スレッドから大域的にアクセス可能なものであって且つスレッド独自に保持されるものとなる。つまり、スレッド固有データを大域変数と同様に宣言すると、これをデータ空間ではなく各スレッドのスタックに多重的に割り当て、各スレッドが自スタックにアクセスすることにより、そのデータはスレッド固有に保持されていることになる。よって、上記の宣言さえ行えば良いのでプログラムが簡素化される。また、固有データのアドレスではなくデータそのものがスタックに保存されているため、実行効率が向上する。さらに、上記の領域確保と基準となるアドレス保持をスレッド生成時にそのスレッドの自スタックについて行うので、スレッドの生成・消滅に伴うスレッド数の動的な変化にも対応することができる。

【0022】

【実施例】以下、本発明の一実施例を図面に従い説明する。図1は同実施例にかかるスレッド固有データ保持方法を行うシステム構成を示すものであり、図4はこの方法を概念的に説明するものである。

【0023】この場合、固有データ認識部3（101）は、コンパイル部2がプログラム入力部1により入力されたプログラム102をコンパイルする際に、このプログラムを解析し、プログラム102中のスレッド固有データを認識し、それらのデータを割り付けるために必要な領域の大きさ、および、その領域内での各データのオフセットを決定するようにしている。この処理を施されたプログラムは実行ファイル103として実行ファイル記憶部4にて記憶される。また、固有データ認識部3（101）は、スレッド固有データ用の領域の大きさを実行ファイル103のヘッダ部に記録し、さらに固有データのオフセットおよび実行時に決定される固有データ領域の開始アドレス（ベースアドレス）により固有データをアクセスする実行コードを実行ファイル103のテキスト部分に記録するようにもしている。以上の動作は、実行前に実行ファイルを作成するにあたって行われるものである。

【0024】上述した固有データ認識部3の動作を図2のフローチャートに示し、これをプログラム102の一例を示す図5を参照しながら説明する。図5のプログラムは、固有データ202、203のようにアクセスする命令を含むｆｕｎｃ1，ｆｕｎｃ2なる関数を実行しようとするものである。図6には、図5に示すプログラムをコンパイルするときの固有データ認識部3の内部データの例を示してある。図7は、図5に示すプログラムをコンパイルして生成される実行ファイル中に、固有データ認識部3が生成して記録する固有データへのアクセス命令の例を示すものである。

【0025】図5において、プログラム中の固有データｄａｔａ1，ｄａｔａ2は固有データであることを示すためにｐｒｉｖａｔｅ宣言（201）されている。すると、固有データ認識部3では、ｐｒｉｖａｔｅというキーワードにより、スレッド固有データを認識する（S1）。そして、図5の場合は固有データが4バイトの整数型の変数2個であるから、固有データ領域（図4の106）の大きさを8バイト（図6の301）と求める（S2）。求めた固有データ領域の大きさを実行ファイル103のヘッダ部分に記録する（S3）。また、各固有データに対してそれぞれ固有データ領域106の内部位置（オフセット）（図6の302）を決定する（S4）。そして、決定したオフセットを基にアクセス命令（図7の401、402）を生成し、これを実行ファイル103のテキスト部分に記録する（S5）。図7に示される命令の意味は、固有データ領域開始アドレス（ＰＳＰ；Private Stack Pointer）から上記のオフセット分離れたアドレスにアクセスせよということである。尚、この例では固有データへの値を代入する命令（図5の202、203）に対応する命令であるから、ｓｔ（store）命令となっている。

【0026】このように作成された実行ファイルの実行時には、実行部5が実行ファイル記憶部4から実行ファイルを読み込み実行を行うが、その際にスレッドを生成したりスレッドを切り替えたりする。スレッドを生成する際のスレッド生成部6（104）の動作を図3（a）に示す。スレッド生成部6（104）は、まず実行ファイル103中のヘッダ部分に記録されたスレッド固有データ用領域の大きさを読み（S11）、この大きさを基に今から生成するスレッドｋのスタック105（7のうちスレッドｋに対応するもの）の最下位部分に固有データ領域106を割り当てる（S12）。そして、これと共に、スレッドの自スタックの最下位アドレスを固有データ領域開始アドレスレジスタ8（107）のＰＳＰのセットする（S13）。さらに、ＰＳＰから上に上記の大きさ分固有データ領域を確保し、このときの上端のアドレスをスタックポインタ9のＳＰ（Stack Pointer）にセットする（S4）。このＳＰは実行時には通常のスタックポインタとして用いられることになる。このようにスレッドを次々と生成していくと、スレッド1～ｎの各スタックにそれぞれ固有データ領域106が割り当てられ、それぞれの開始アドレスがスレッド毎のＰＳＰのセットされることになる。この様子を図8に示す。尚、ＰＳＰを各スタックの最下位アドレスとしたのは便宜上であり、各スタック中の任意の位置の固有データ用の領域を設ける基準アドレスと置き換えても良い。

【0027】このように設けられた固有データ領域中の固有データへのアクセスの実行は、上述したようにコンパイル時に固有データ認識部3が生成した実行ファイル中のコード401、402の中のＰＳＰのところに、現在のスレッドの固有データ領域開始アドレスレジスタ8（107）に格納されたアドレスの値を代入し、この値

とコード中のオフセットの値とで計算されるアドレスにアクセスすることにより行われる。

【0028】一方、実行時にスレッドを切り替える際のスレッド切替え部10（108）の動作を図3（b）に示す。スレッド切替え部10（108）は、スレッド切替えを行う際に、現在のPSPを他のレジスタ（SP等）と共に読み出して（S21）これらの値をスレッドコンテキスト管理部10に保存する（S22）。スレッドコンテキスト管理部10には、スレッド生成時にセットされた固有データ領域の開始アドレス、SP等をスレッド毎に保持してある。そこでスレッド切替え部10は、このスレッドコンテキスト管理部10から次に実行するスレッドのPSP、SPを読み出して（S23）これらの値を固有データ領域開始アドレスレジスタ8、スタックポインタ9にセットする（24）。

【0029】このような操作により、常に、あるスレッドの実行中はそのスレッドの固有データ領域の開始アドレスがPSPにセットされていることになり、固有データへのアクセス命令は実行中のスレッドの固有データ領域に対して行われるようになり、現在実行されていないスレッドの固有データ領域へのアクセスは行われないことになる。

【0030】

【発明の効果】本発明によれば、スレッドに固有な大域データは、一般の大域データと同様に記述することができるようになる。このため、Auto変数として宣言して記述した時に問題となるプログラムの繁雑さを防ぐことができ、プログラムの生成が向上するだけでなく、バグの混入を防止する上でも大きな効果がある。また、固有データは、静的に固有データ領域中の位置が定められるため、アドレスによる間接アクセスの必要がないことと、関数呼び出しごとにそのアドレスを引き渡す必要がないため、実行効率を向上させることができる。さらに、固有データ領域を各スレッドのスタック中に確保するため、スレッド数に制限がなく、大域的な配列として宣言する場合と比べてスレッドの動的生成・消滅に容易に対応できることになる。このように実用上多大なる効果を奏するスレッド固有データ保持方法が実現できる。

【図面の簡単な説明】

【図1】本発明の一実施例に係るスレッド固有データ保持方法を行うシステム構成を表わすブロック図。

【図2】図1の固有データ認識部3の動作を示すフローチャート。

【図3】図1の実行部5が（a）スレッド生成部6を起動した場合の6の動作、（b）スレッド切替部10を起動した場合の10の動作を示すフローチャート。

【図4】本発明の一実施例に係るスレッド固有データ保持方法を概念的に示す図。

【図5】本実施例に用いられるプログラム例を示す図。

【図6】図5に示すプログラムを解析した場合の固有データ解析部の内部データを示す図。

【図7】図5に示すプログラムをコンパイルした実行ファイルを示す図。

【図8】実行中のスレッドのスタック状態を示す図。

【図9】従来のスレッド固有データを保持する方法のプログラム例を示す図。

【図10】従来のスレッド固有データを保持する方法のプログラム例を示す図。

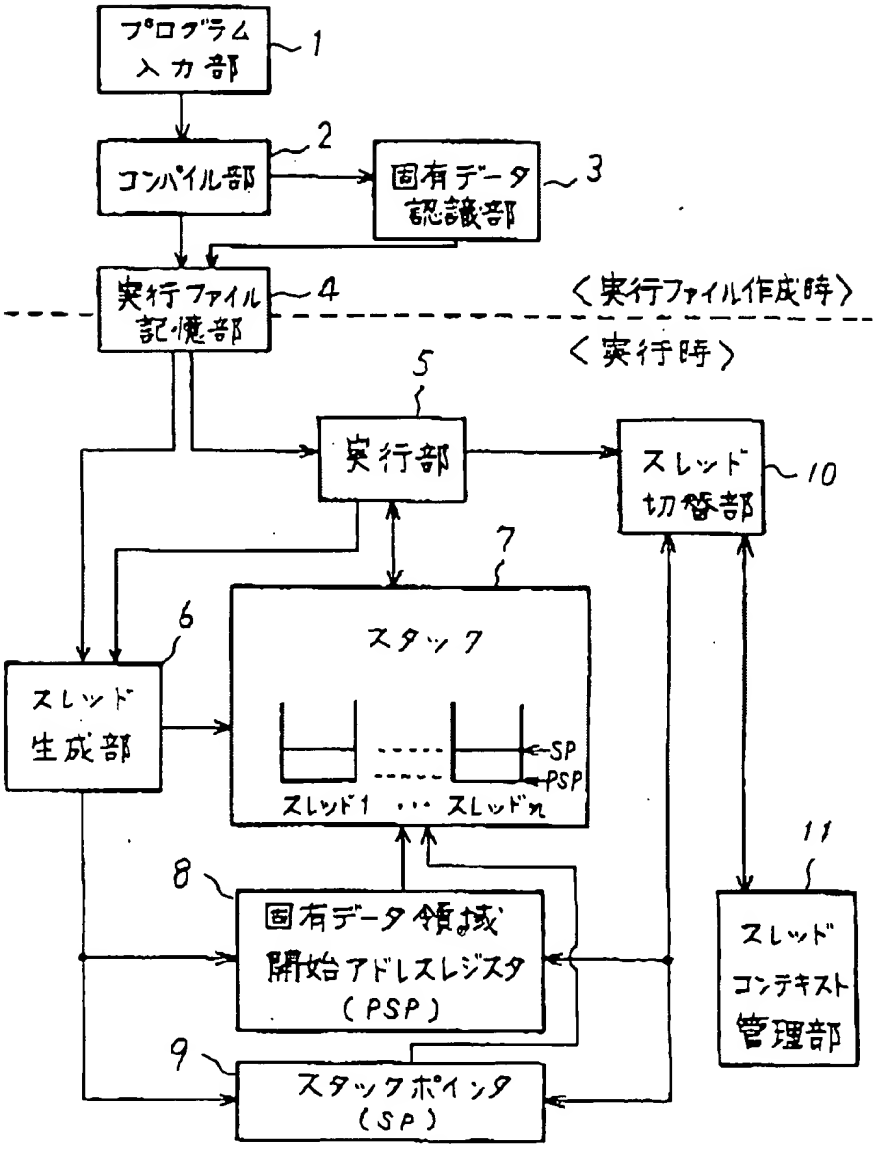
【符号の説明】

1…プログラム入力部、2…コンパイル部、3，101…固有データ認識部、4…実行ファイル記憶部、5…実行部、6，104…スレッド生成部、7，105…スタック、8，107…固有データ領域開始アドレスレジスタ、9…スタックポインタ、10，108…スレッド切替え部、11…スレッドコンテキスト管理部、102…プログラム、103…実行ファイル、106…固有データ領域。

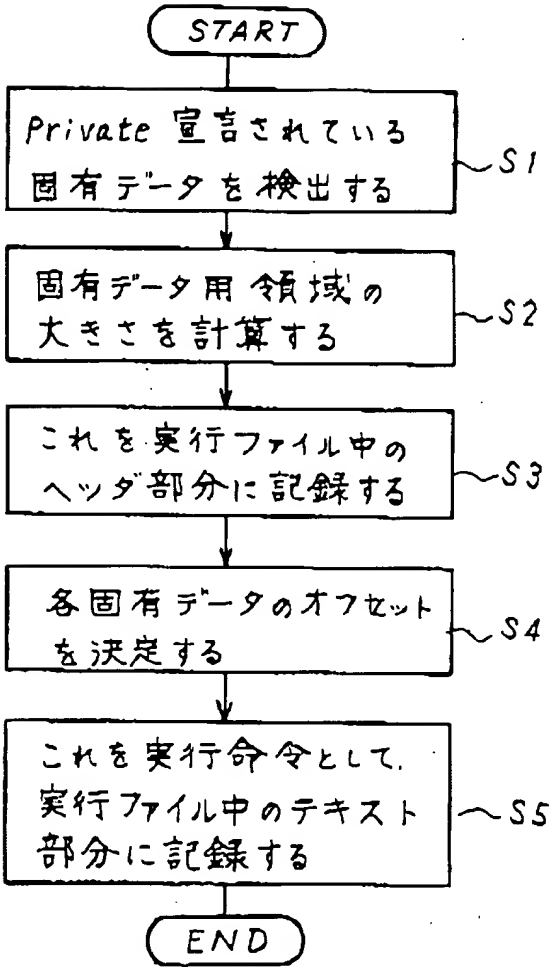
【図6】

固有データ領域サイズ＝8バイト	～301
固有データの位置	
名前	位置 ～302
data1	0
data2	4

【図1】

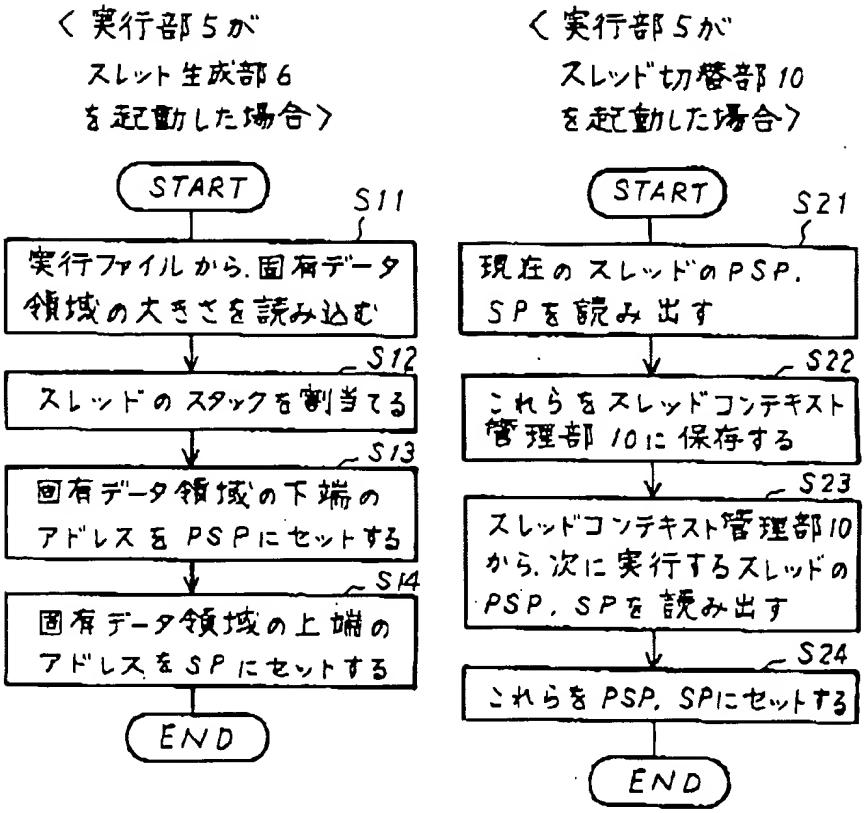

プログラム入力部
1
コンパイル部
2
固有データ認識部
3
実行ファイル記憶部
4
〈実行ファイル作成時〉
〈実行時〉
5
実行部
スレッド切替部
10
7
スタック
6
スレッド生成部
SP
PSP
スレッド1 ・・・ スレッドn
11
8
固有データ領域開始アドレスレジスタ（PSP）
スレッドコンテキスト管理部
9
スタックポインタ（SP）

【図2】



〈固有データ認識部3の動作〉

【図3】



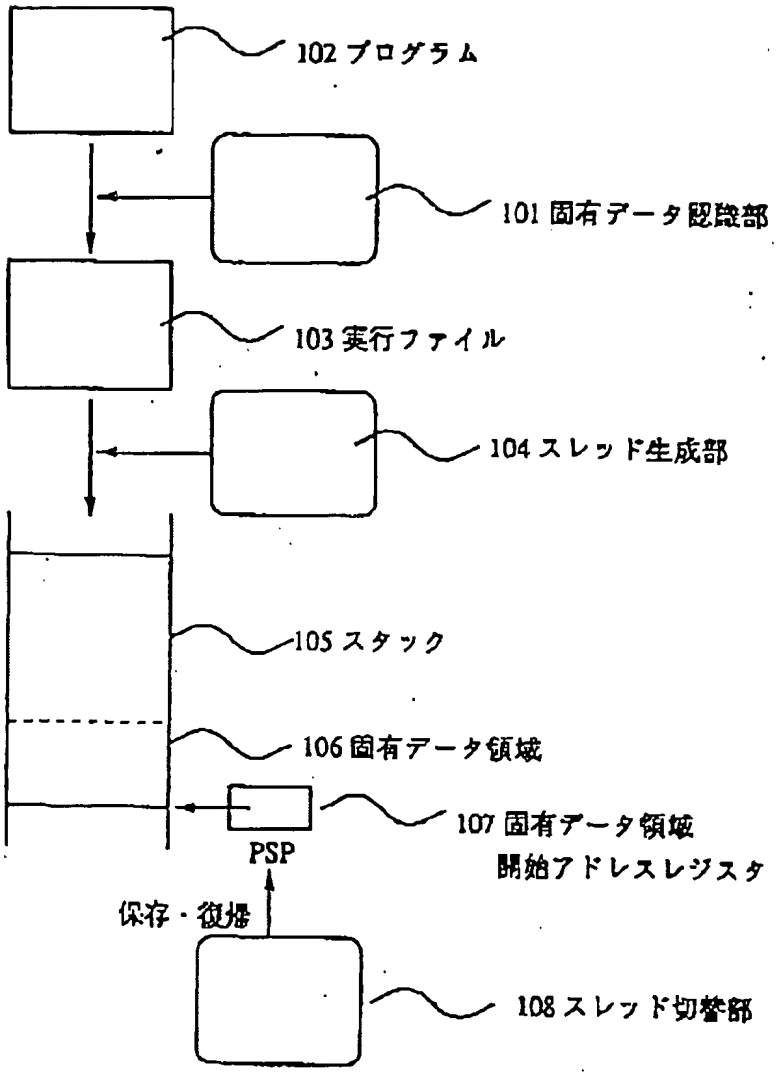
【図5】

```
private int data1, data2;        ~ 201

func1(){
    data1 = X;                   ~ 202
}

func2(){
    data2 = Y;                   ~ 203
}
```

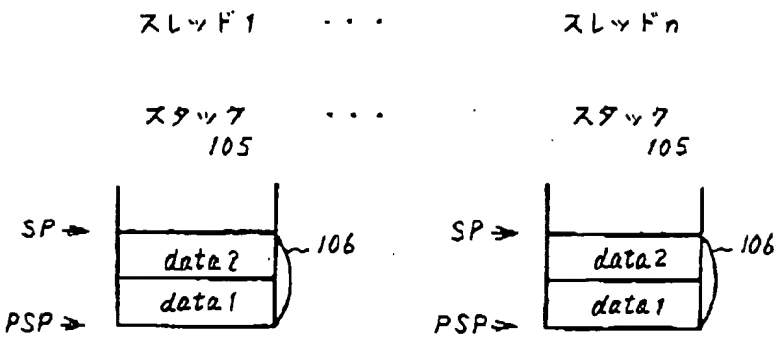
【図4】


102 プログラム
101 固有データ認識部
103 実行ファイル
104 スレッド生成部
105 スタック
106 固有データ領域
107 固有データ領域
開始アドレスレジスタ
PSP
保存・復帰
108 スレッド切替部

【図7】

```
func1:
    ・・・
    st(PSP-0), x ～ 401
    ・・・
func2:
    ・・・
    st[PSP-4], Y ～ 402
    ・・・
```

【図8】



【図9】

```
start()
{
    int shared_data;

    ・・・ = func1(&shared_data);
    ・・・ = func2(&shared_data);
}

func1(ptr1)
    int *ptr;
{
    ・・・
    *ptr = X;
    ・・・
}

func2(ptr)
    int ptr;
{
    ・・・
    Y = *ptr;
    ・・・
}
```

【図10】

```
int shared_data[THREAD_NUM];

start()
{
    ・・・ = func1();
    ・・・ = func2();
}

func1()
{
    shared_data[THREAD_SELF] = X;
}

func2()
{
    Y = shared_data[THREAD_SELF];
}
```